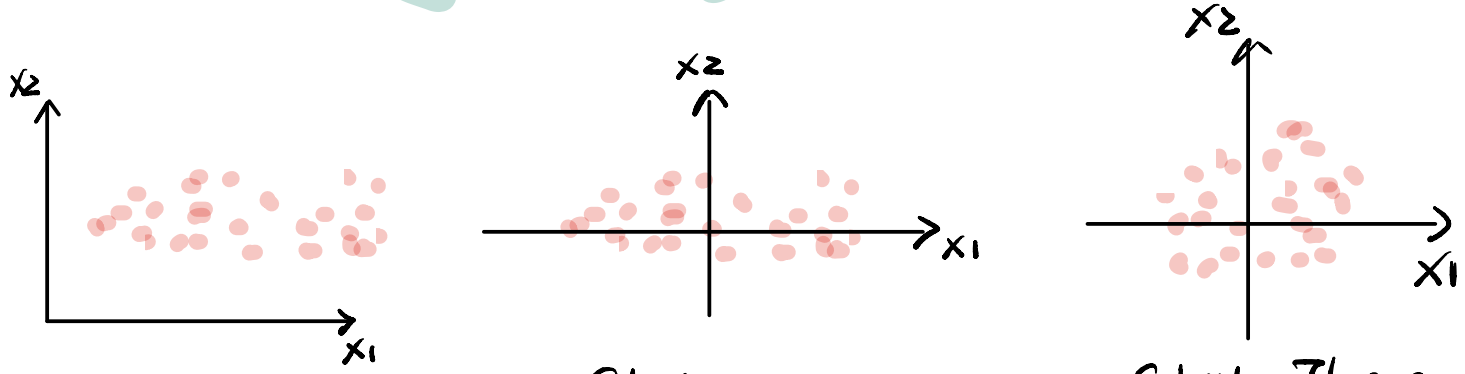
Normalizing training sets



State One State Two State Three.

Step one: Subtract Mean (State 1 → State 2).

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}.$$

$$x := x - \mu$$

Effect: Centering data around mean.

Step two: Normalize variance (Stage 2 → Stage 3).

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} ** 2$$

$x /= \sigma$: Subtract mean and divided by Standard deviation.

* Same σ and μ need to be applied to normalize test set.

Why normalize input?

w_1: x_1: 1 – 1000

w_2: x_2: 1 – 2

Since $J(w,b) = \frac{1}{m}\sum_{i}^{m} L(\hat{y}, y)$, and $\hat{y} = w_1x_1 + w_2x_2 + \cdots w_nx_n + b$, w_1 in example on the left will have bigger effect on our result than w_2, which is an assumption that we don't want to make.

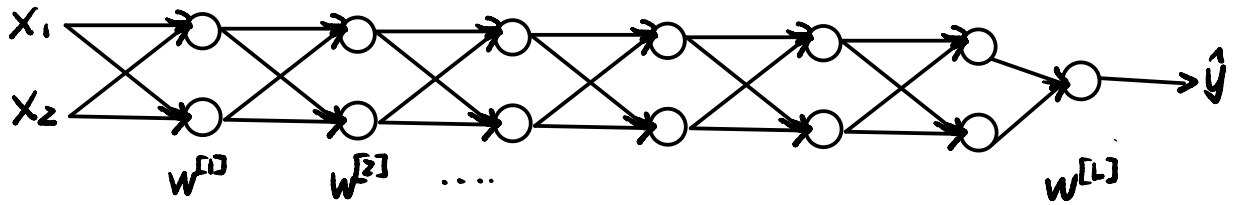
Another reason for normalizing inputs:

Neural Networks with Unnormalized inputs have more oval cost function shape comparing a more circular cost function resulting from normalized neural networks. As a result, gradient descent will do more zigzaging movements and increase training expense in unormalized networks.

In short: Unnormalized inputs = harder to learn thru gradient descent.

vanishing / exploding gradients

Imagine a very deep neural network:



Assume $g(z) = z$, $b^{[l]} = 0$

$\rightarrow \hat{y} = w^{[L]} w^{[L-1]} \cdots w^{[1]} x$

if $w^{[l]} > I$, e.g. $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

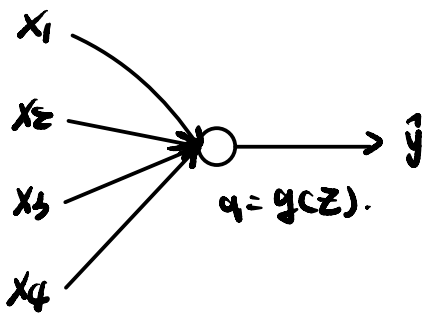
$\hat{y} = (w^{[l]})^L x \Rightarrow \hat{y}$ and da will explode.

if $w^{[l]} < I$, e.g. $w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

$\hat{y} = (w^{[l]})^L x \Rightarrow \hat{y}$ and da will vanish.

Solution for exploding / vanishing problem.

Single neuron example:



$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots w_n x_n + b$ (b ignored for now.)

Intuition: large n → smaller w_i. So that z won't explode or vanish.

Thus: $Var(w_i) = \frac{1}{n} \Rightarrow Var(n w_i) = 1$

Prevent w_i from being too big or too small.

Implementation:

$Var(W) \approx 1 \sim 2$.

$w^{[l]} = np.random.randn(shape) * np.sqrt(\frac{2}{n^{[l-1]}})$

mean at 0

$\frac{2}{n}$ rather than $\frac{1}{n}$

Works best for Relu.

Other variants:

tanh:

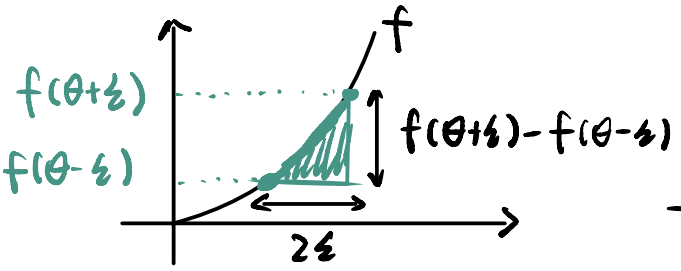
$\sqrt{\frac{1}{n^{[l-1]}}}$

xavier initialization:

$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Gradient Checking

Numerical approximation of gradients



$$\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\rightarrow f(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon} \rightarrow O(\varepsilon^2)$$

Error rate.

Gradient checking for a neural network.

- Take $W^{[1]}, b^{[1]} \dots W^{[L]}, b^{[L]}$ and reshape into a big vector θ.
- Take $dW^{[1]}, db^{[1]} \dots dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

Grad check:

for each i:

$$\rightarrow d\theta_{approx}^{[i]} = \frac{J(\theta_1, \theta_2 \dots \theta_i+\varepsilon, \dots) - J(\theta_1, \theta_2 \dots \theta_i-\varepsilon \dots)}{2\varepsilon}$$

$$\approx d\theta^{[i]} = \frac{\partial J}{\partial \theta_i} \quad \Big| \quad d\theta_{approx} \overset{?}{\approx} d\theta.$$

Check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \approx$

10^{-7} : great

10^{-5} : ok

10^{-3} : worry

$\varepsilon = 10^{-7}$

Gradient Checking Implementation Notes:

- Don't use it in training - only to debug
- if grad check fails, check components with biggest difference to identify bug.
- Remember the effect of regularization
- Doesn't work with dropout
- Run with random initialization. perhaps again after some training.